Even though classification models are not trained for it, they can also be used to generate new synthetic data points. One technique is to randomly generate samples within the input feature space and use the model to classify them. The sample can be considered a synthetic data point for the class it was predicted to be. This method can eventually enumerate all the examples within a given class if given infinite time, but generally this method is less than practical.

Another technique is to use iterative backpropagation to refine inputs to a desired class. For example, using the MNIST dataset, you can start with an input of random noise and backpropagate the error not through to the weights, but all the way through to the inputs. This gradient allows for modifying the input to be more like the desired class. In practice, this approach typically leads to adversarial examples that strongly activate the desired class but don't particularly resemble real data.

While networks trained for classification can theoretically be used for generation through techniques like random sampling and activation maximization, there are severe limitations because classification models are designed to map inputs to discrete categories, not learn the underlying distribution of the data.

With that being said, there are some distinct modifications that would make the architecture better suited to generating new data. First, the model outputs need to be changed from class probabilities to continuous outputs for each input feature (columns if tabular, pixels for images). The loss function would also need to be modified to optimize reconstruction quality rather than classification accuracy. An additional modification that could improve generation performance is adding random noise to samples. Using noisy samples can improve generalization since the model can learn to be invariant to small changes that don't alter the validity of the sample (similar to denoising autoencoders).

Extending this idea are generative architectures such as autoencoders and generative adversarial networks (GANs). Autoencoders work by attempting to reconstruct an input after passing it through a bottleneck that forces the model to compress the information to an efficient representation that takes advantage of patterns in the input data. After training, new synthetic data points can be generated by sampling randomly (or semantically) from the latent space and passing it through the decoder.

On the other hand, GANs work by combining two separate models in an adversarial fashion. Specifically, they consist of generator and discriminator networks. The discriminator attempts to correctly distinguish real samples from fake samples and the generator tries to create realistic fake samples.

In the context of wine quality, we can use our existing classification model with few modifications as the discriminator in a GAN system. Instead of predicting the specific rating given by the wine reviewers, the model can take in all the variables (including the score) and produce a scaler probability of whether the sample is real or not. For our generator, we can use a CNN model that has the same output dimensions as the input to the generator. Then, we can use the traditional GAN training formula to fit our model to the data and hopefully produce high quality synthetic samples.